7 Using Engine Manager

Overview

Engine Manager is composed of the file enmgr.dll and support programs and files. It is an integral part of DSPower-HWLib and DSPower-SPLib software. Its main purpose is to provide control over DSP Engines, so that they can be opened and perform useful work more or less transparently to the calling application. Engine Manager can keep track of multiple engines, including multiple copies of the same engine, at one time. It generates callback messages when necessary, redirecting information to previously designated user window procedures. Engine Manager contains internal GetMessage loops, which are used when a synchronous call has been made and it must wait before returning results to a calling application.

For a concise description and explanation of individual calls inside Engine Manager see Signalogic DSP Software Reference Guide section 2.2.1, Engine Manager Functions.

For more information about, and the basic philosophy underlying the engine concept, see DSP Engines, in Signalogic DSP Software System Guide section 1.1, Relationship of Signalogic DSP Software Components.

Main Topics

Engine Manager Access
Controlling DSP Engines
Executing Functions in DSP Engines
Registering Callback Messages from DSP Engines
Error Handling in Engine Manager
Error/Status Window in Engine Manager

7.1 Engine Manager Access

[@k implicit loading, explicit loading]

To access Engine Manager, the enmgr.dll library must be visible to your application's calling program. The easiest way to do this is to include the enmgr.lib file in your project. For C/C++ projects, please see the example Borland C/C++ v3.1 and v4.5, and the Microsoft Visual C/C++ v1.5 and v2.0, project files, included with the DSPower-HWLib or DSPower-SPLib software. For MATLAB projects, please see the example .m files. Once included in your project file, and thus linked into your final executable, the enmgr.lib file causes the Engine Manager DLL to be loaded and initialized when your executable first runs (actually, *before* any code in the executable runs). This is known as implicit loading.

Another way of accessing the enmgr.dll library is to load it from your application program dynamically, using the LoadLibrary call in the Windows API, followed by one or more GetProcAddress calls with specific function names as input parameters. This method is known as explicit loading, and is more complicated, requiring painstaking typecasting to create function prototype *pointers* which are set as a result of GetProcAddress calls. "Behind the scenes", this is also the technique used by MATLAB to load DLLs needed by calling .m programs--except that MATLAB simplifies the process by allowing DLL calls to look much like other MATLAB function calls.

Also, for C/C++ programs, the enmgr.h include file must be specified along with other include files that may be needed by the calling application. Note that the enmgr.h include file expects the standard Windows 3.1 windows.h include file to have been previously specified.

NOTE: For C/C++ development, make sure the include file directory list in your IDE contains the DSPower-HWLib subdirectory. The example project files supplied with DSPower-HWLib software assume default include file subdirectories of "hwlib", "toollib", and "helplib" under the root-level installation subdirectory, although toollib and helplib are present only for demo purposes.

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

Error/Status Window in Engine Manager

7.2 Controlling DSP Engines

The main purpose of Engine Manager is to abstract the control of diverse DOS and Windows applications needed to perform DSP/analog hardware- or signal processing-related work. The core of the Engine Manager API consists of the following calls:

- DSEngineOpen
- DSEngineClose
- DSGetEngineUsage
- DSExecEngineFunction
- DSSendEngineCommand
- DSSetEngineWaveformPath

- DSSetEngineTemplatePath
- DSGetEngineErrorStatus
- DSRegisterEngineMsgWnd

For detailed information on these and other calls inside the Engine Manager API, including input parameters and output results, see Signalogic DSP Software Reference Guide section 2.2.1, Engine Manager Functions.

Related Topics

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

Error/Status Window in Engine Manager

7.2.1 Opening DSP Engines

Before many of the functions in DSPower-HWLib and DSPower-SPLib DLLs can be used, a DSP Engine must first be opened. In the source code example programs included with DSPower-HWLib and DSPower-SPLib software, a DSEngineOpen call is usually one of the first few statements. Often, this call is made and the program has little further need to directly use the engine handle returned. Typical uses of the engine handle are:

- making DSAssignBoard calls in DSPower-HWLib
- executing DSP Engine functions using the DSExecEngineFunction call (see section 7.3 below, Executing Arbitrary Functions in DSP Engines)
- registering callback windows for various purposes

- closing engines using DSEngineClose (which is also done automatically by enmgr.dll when user-defined programs terminate, if needed)

The DSEngineOpen call opens DSP Engines such as Hypersignal-Macro series programs or MATLAB programs and places them in command mode, ready to accept instructions. DSP Engines are opened so as not to be visible (and conserve memory by not loading graphics drivers or other user-interface objects, if possible). The form of the call is as follows:

```
hEngine = DSEngineOpen(szEnginePath, szWaveformPath);
```

where szEnginePath and szWaveformPath are null-terminated strings (both input parameters are defined as LPCSTR). The return value, hEngine, is non-NULL on success, and NULL on failure. If the call fails, the DSGetEngMgrErrorStatus call can be used to get more information (see also section 7.5 below, Error Handling in Engine Manager). szEnginePath can be set to one of the following pre-defined constants to conveniently specify supported DSP Engines:

szEnginePath	**DSP Engine**	**Default Search Path**
DS_EO_HSMNRT	Hypersignal-Macro NRT	x:\hsmacro\hsmacnrt.exe
DS_EO_HSM	Hypersignal-Macro RT	x:\hsmacro\hsmacro.exe
DS_EO_HSA	Hypersignal-Acoustic	x:\hsacous\hsacous.exe
DS_EO_MAT	MATLAB v4.x	x:\matlab\bin\matlab.exe

If one of the above constants is used, then the default subdirectories shown are used to attempt to locate the engine. For example, MATLAB is searched for as "x:\matlab\bin\matlab.exe", where "x" is the currently active drive. If instead the szEnginePath parameter contains a pointer to a valid string, then this string is assumed to contain explicit path information for the DSP Engine.

If the engine path is left NULL, then the DSEngineOpen function attempts to locate an appropriate DSP Engine, first trying Hypersignal-Macro software, then Hypersignal-Acoustic software, etc. This approach is used in the DSPower-HWLib demo programs, so these programs will run with whatever engine is installed on the system.

The szWaveformPath parameter can be set when a DSP Engine is open, or at any time using the DSSetEngineWaveformPath call. This path specifies where the DSP Engine will look when processing waveform files.

Internally, DSP Engines are opened and initialized using either a Windows WinExec call or functions inside shell.dll. If the WinExec call is used, the "show state" parameter is set to SW_HIDE, and in the case of DOS engines, a .pif file may be associated with the engine.

For more information, see also DSEngineOpen in Signalogic DSP Software Reference Guide section 2.2.1, Engine Manager Functions.

Related Topics

Sending DSP Engine Commands

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

Error/Status Window in Engine Manager

7.2.2 Sending DSP Engine Commands

From time to time, user-defined programs may need to send single commands or a stream of commands to an open DSP Engine. Being able to do so offers ultimate flexibility and may be the "last line of defense" in handling unusual situations. The DSSendEngineCommand call in enmgr.dll is intended for this purpose. For example, to abort a MATLAB function taking a long time to process, you can stop the function using:

```
DSSendEngineCommand(hEngine, 3);
```

This sends the ctrl-C command to MATLAB. As another example, you can stop and restart an executing Hypersignal-Macro series software function without the overhead of submitting a new macro language source file by using:

```
DSSendEngineCommand(hEngine, 0x1b);
DSSendEngineCommand(hEngine, 0x0a);
```

The 0x1b value is equivalent to pressing the Escape key, and the 0x0a value is equivalent to the Ctrl-Enter key sequence. The above example might apply in a situation where you had changed a waveform file or filter coefficient file needed by the Hypersignal function, and then wanted to stop and restart the function causing it to use the new information.

Related Topics

Opening DSP Engines

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

Error/Status Window in Engine Manager

7.3 Executing Functions in DSP Engines

Engine Manager has a general-purpose DSExecEngineFunction function that allows DSP Engine functions to be invoked from user-defined programs. This facility is not to be confused with DSP Engine functions, or combinations of functions, which are executed automatically and transparently as a result of standard API calls inside DSPower-HWLib or DSPower-SPLib DLLs.

The purpose of the DSExecEngineFunction call is to allow engine functions to be called directly, assuming knowledge of the syntax and parameters of the engine call. Input parameters are the engine handle and a string, which should define the complete engine command. For example, to evaluate a difference equation on a waveform file with Hypersignal-Macro as the engine, a statement like the following might be used:

```
DSExecEngineFunction(hEngine, "f_diffeq(\"sp_avg\", \"y[n] = y[n] + (x1[n] – x1[n-M])/Fr\", \"speech\", \"\", 0, 0, 0)", uMode);
```

Note the use of \" in the above calls when quote marks are needed inside a C/C++ language string already differentiated by quote marks. Note also the hEngine parameter, which should represent a valid engine handle from a previous DSEngineOpen call, and the uMode parameter, which specifies either synchronous or asynchronous operation. Synchronous operation means the calling application waits for the engine function to complete; asynchronous operation means the calling application continues to execute simultaneously. uMode values of DS_EEF_SYNC and DS_EEF_ASYNC are predefined inside the enmgr.h include file.

Currently, engine functions cannot be queued via repeated calls to DSExecEngineFunction. However, this same functionality can be achieved by queuing calls in a local structure defined and maintained in the calling application, and then by emptying the queue in response to program done messages. See Program Status Messages from DSP Engines, in section 7.4 below, Registering Callback Messages from DSP Engines.

As another example, to call a simple FFT inside MATLAB, a statement like the following might be used:

```
DSExecEngineFunction(hEngine, "x = f_load(\"speech.wav\")", DS_EEF_SYNC);
DSExecEngineFunction(hEngine, "y = fft(x)", DS_EEF_ASYNC);
```

Note that some method must be utilized to initialize arrays before being used as input to MATLAB functions. The above example shows the DSPower series software f_load function being used to load the time domain waveform file "speech.wav". Note also that the first function has been executed synchronously, which means the function does not return until complete. The second function has been executed asynchronously, which means that the calling application continues to execute after the function has been started.

NOTE: If a callback window has been registered using the DSRegisterEngineMsgWnd function, a "program done" message will be sent to this window when the engine function specified in a DSExecEngineFunction call completes. See Program Status Messages from DSP Engines, in section 7.4 below, Registering Callback Messages from DSP Engines.

For more information on DSPower series software functions, see Signalogic DSP Software Reference Guide section 2.1, DSP Engine Functions. For more information on MATLAB functions, see the MATLAB Reference Guide and the Signal Processing Users Guide (The MathWorks).

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

Error/Status Window in Engine Manager

7.4 Registering Callback Messages from DSP Engines

Engine Manager allows user-specified window or control handles to be "registered" to several types of DSP Engine callback messages, using the DSRegisterEngineMsgWnd function. Once registered, window or control handles receive callback messages at various times, depending on various circumstances.

Below are the current callback message types available in DSPower-HWLib and DSPower-SPLib software:

- engine status messages

- program status messages
- buffer ready messages.
- semaphore messages
- statistics and parameter update messages

A window or control handle must be registered for each of the above callback message types before they can be sent by DSP Engines. Engine and program status messages are handled by Engine Manager (enmgr.dll), and buffer ready, semaphore, and statistics and parameter update messages are handled by either DSPower-HWLib (hwlib.dll) or DSPower-SPLib (splib.dll).

Below are some examples of using the DSRegisterEngineMsgWnd function:

```
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETSTATISTICSMSG |
DS_REMW_SETLEVELMSG, hwndCtl);
```

In the above call, the DS_REMW_SETSTATISTICSMSG and DS_REMW_SETLEVELMSG constants have been used to indicate that amplitude level and statistics and parameter update messages should be sent to hwndCtl. For instance, hwndCtl could be a static control in a dialog box used to display text information, or a graphical meter control. Both of these examples plus more are shown in the source code file for the digital tape recorder demo included with DSPower-HWLib software (dtape.cpp).

Note in the above example that callback message type constants can be combined. There is also a DS_REMW_SETALLMSG constant which can be used to register one window for all callback message types. Below is another example, which would apply in the case of executing general engine functions with the DSExecEngineFunction call:

```
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETPROGSTATUSMSG,
hwnd);
```

In the above call, the DS_REMW_SETPROGSTATUSMSG constant has been used to indicate that program status messages should be sent to hwnd. For example, if the DSExecEngineFunction call was to be used to start an engine function or program, then the calling application might first specify that hwnd should receive a WM_DSPENGINE_PROGSTATUS message indicating completion of the function or program.

Below is an example which could be used in the case of processing continuous analog input data from DSP/analog hardware:

```
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG,
hwnd);
```

In the above call, the DS_REMW_SETDSPDATARDYMSG constant has been used to indicate that buffer ready and semaphore ready messages available from DSP Engines via DSPower-HWLib should be sent to hwnd. See Example C/C++ Code: Processing Buffer Ready Messages in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software, for a complete example showing a typical response to buffer ready messages.

Below are the possible callback message types for which window or control handles can be registered:

Update Message Category	**Summary of Meaning**
DS_REMW_SETENGINESTATUSMSG	Register callback handle to receive all engine-related status messages.
DS_REMW_SETPROGSTATUSMSG	Register callback handle to receive program and function status messages (caused by invoking high-level functions such as DSExecEngineFunction or DSAcquireWvfrmFile).
DS_REMW_SETDSPDATARDYMSG	Register callback handle to receive buffer ready and semaphore ready messages.
DS_REMW_SETSTATISTICSMSG	Register callback handle to receive DSP/analog hardware statistic and parameter messages.
DS_REMW_SETBUFNUMMSG	Register callback handle to receive buffer number messages.
DS_REMW_SETLEVELMSG	Register callback handle to receive amplitude level messages.

DS_REMW_SETALLMSG	Register callback handle to receive all of above messages.

For the most complete source code examples of registering and processing the above callback message types, please see the dtape.cpp C/C++ file included with DSPower-HWLib software.

Callback Message Types

Status Messages from DSP Engines

Program Status Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

Update Messages from DSP Engines

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Processing Buffer Ready Messages

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

DSPower-HWLib Function Overview

Creating Programs for DSPower-HWLib and DSPower-SPLib

Acquiring Data Using DSPower-HWLib Software

Status Messages from DSP Engines

Engine status messages are sent by DSP Engines via Engine Manager regarding general operations and error conditions. No messages are sent unless user-specified window or control handles are registered using the DSRegisterEngineMsgWnd function. Below are the current engine status messages possible:

Engine Status Message	Summary of Meaning	wParam	lParam
WM_DSPENGINE_ENGINESTATE	Engine state has changed (for example, engine has returned to idle state).	engine identifier	low word has current engine state
WM_DSPENGINE_CODEGENERROR	Engine encountered a format error or an error in locating source or executable program file.	engine identifier	
WM_DSPENGINE_COMPILEERROR	Engine encountered an error when compiling a native language source file.	engine identifier	pointer to error informa-tion
WM_DSPENGINE_FUNCTIONERROR	Engine function encountered an input or setup error (before function could start).	engine identifier	
WM_DSPENGINE_RUNTIMEERROR	Engine encountered a run-time error (for example, waveform or other data file error, DSP/analog hardware error, etc.).	engine identifier	pointer to error informa-tion

The lParam parameter in the error messages listed above points to an ERRORMSGINFO structure, which contains information about the error, including error number and description, error log filename, etc. The calling application does not need to return a specific value after processing engine status messages.

For detailed information about the above messages, see Signalogic DSP Software Reference Guide section 6.1, DLL Messages.

See the enmgr.h file for the current actual value of these messages. Normally, program status messages are defined as WM_USER + n,

where n is a constant and WM_USER is defined in windows.h. Other WM_DSPENGINE messages are defined similarly.

Callback Message Types

Status Messages from DSP Engines

Program Status Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

Update Messages from DSP Engines

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Processing Buffer Ready Messages

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

DSPower-HWLib Function Overview

Creating Programs for DSPower-HWLib and DSPower-SPLib

Acquiring Data Using DSPower-HWLib Software

Program Status Messages from DSP Engines

Program status messages are sent by DSP Engines as a result of the calling application invoking a high-level function, such as DSExecEngineFunction or DSAcquireWvfrmFile. Messages are only sent if the calling application has previously registered a window or control handle using the DSRegisterEngineMsgWnd function.

Current program status messages in DSPower-HWLib and DSPower-SPLib software are listed below:

Program Status Message	Summary of Meaning	wParam	lParam
WM_DSPENGINE_PROGSTATUS	Program (or functions within program) started by DSExecEngine-Function completed.	engine identifier	low word indicates function or program

For detailed information about the above messages, see Signalogic DSP Software Reference Guide section 6.1, DLL Messages.

See the enmgr.h file for the current actual value of these messages. Normally, program status messages are defined as WM_USER + n, where n is a constant and WM_USER is defined in windows.h. Other WM_DSPENGINE messages are defined similarly.

Callback Message Types

Status Messages from DSP Engines

Program Status Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

Update Messages from DSP Engines

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Processing Buffer Ready Messages

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

DSPower-HWLib Function Overview

Creating Programs for DSPower-HWLib and DSPower-SPLib

Acquiring Data Using DSPower-HWLib Software

Buffer Ready Messages from DSP Engines

Buffer ready messages are sent by DSP Engines when a pre-specified buffer number is set, indicating that another analog input buffer has been acquired or another analog output data buffer is needed from the calling application. Reserved values can be used to indicate that buffer ready messages should be generated by a DSP Engine as a result of native acquisition, digital oscilloscope, or other operating modes which have been initialized and started by the calling application. They can also be generated by user-defined DSP code using either polled or interrupt-driven notification. An example of the former might be buffer switching for continuous analog input or output transfer between disk file.

Buffer ready messages are set up when an application calls DSWaitForBuffer. Below is a source code example:

```
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG, hwnd);
DSWaitForBuffer(hBoard, nBufferNum, dwFlagAddr, uNotifyType);
```

In the above calls, hwnd is first registered to receive buffer ready and semaphore ready messages from the DSP Engine; in this case, hwnd will receive WM_DSPENGINE_BUFRDY messages (see below). hBoard is a handle to a DSP/analog hardware type created by DSAssignBoard, nBufferNum is the buffer number to wait for, dwFlagAddr is the address in DSP memory containing the buffer number, and uNotifyType is the type of notification for the engine to use.

nBufferNum values 0 through 15 are reserved for standard DSP Engine high-level function "data nodes", which are pre-defined sources and sinks of analog input and output data. In these cases, dwFlagAddr and uNotifyType are ignored. For example, nBufferNum values of 0 and 1 indicate continuous analog input data, alternating between buffers 0 and 1. uNotifyType is ignored for reserved nBufferNum values because DSP Engines may use specialized methods of buffer-checking, which may involve hardware register bits or other optimized techniques (not necessarily DSP board memory). The concept behind reserved data nodes is that more than one calling application may register a window handle to receive the same buffer ready messages.

Further defined are subpairs of reserved nBufferNum values corresponding to nodes which use a dual-buffer switching mechanism for processing continuous data. nBufferNum values of 0 and 1 indicate continuous analog input data, alternating between buffers 0 and 1, nBufferNum values of 2 and 3 indicate continuous analog output data, values of 4 and 5 indicate telephone adapter input data, etc. For example, if operating Hypersignal-Macro series software in its digital oscilloscope mode, then the first DSWaitForBuffer call would specify buffer 0. After receiving this buffer, another DSWaitForBuffer call would be issued with buffer = 1, then again with buffer = 0, etc.

Values for nBufferNum not equal to 0 through 15 can be used to set up user-defined buffer ready messages. In these cases, dwFlagAddr and uNotifyType must be specified. Note that if the initial value at dwFlagAddr should not be zero, then a DSPutMem call should be performed before DSWaitForBuffer to initialize the flag value correctly.

NOTE: In the case of user-defined buffer ready messages, buffer numbers and memory flag locations used must be "agreed to" beforehand by the calling application and the executing DSP code.

Currently, values for uNotifyType can be DS_WFB_POLLED and DS_WFB_INT, which specify polling and interrupt-driven notification, respectively. If a uNotifyType value of DS_WFB_POLLED is specified, then the DSP Engine will periodically check to see if the specified buffer is ready, and cause WM_DSPENGINE_BUFRDY messages to be sent to hwnd as needed. Currently, if uNotifyType is DS_WFB_POLLED, then only one DSWaitForBuffer call can be active at one time for each active DSP Engine. To establish multiple DSWaitForBuffer requests at one time in polled mode, more than one DSP Engine handle should be active; i.e. more than one copy of .the DSP Engine open simultaneously.

If a uNotifyType value of DS_WFB_INT is specified, the DSP Engine will set up and enable the next available DSP-to-host PC interrupt. When the interrupt occurs, a WM_DSPENGINE_BUFRDY message is sent to hwnd. Note that if uNotifyType is DS_WFB_INT, then more than one DSWaitForBuffer call can be established at one time for each active DSP Engine. Note also that for user-defined buffers using DS_WFB_INT, the dwFlagAddr location is checked to make sure it has changed from its initial value.

Current buffer ready messages in DSPower-HWLib and DSPower-SPLib software are listed below:

Buffer Ready Message	**Summary of Meaning**	**wParam**	**lParam**
WM_DSPENGINE_BUFRDY	DSP data buffer is ready	engine identifier	pointer to buffer inform-ation

The lParam parameter points to a DSPDATARDYMSGINFO structure, which contains information about the buffer, including buffer number, board handle, and flag address and value (if any). The calling application does not need to return a specific value after processing buffer ready messages.

For detailed information about the above messages, see Signalogic DSP Software Reference Guide section 6.1, DLL Messages.

See the enmgr.h file for the current actual value of these messages. Normally, WM_DSPENGINE_BUFRDY is defined as WM_USER (defined in windows.h), and subsequent messages are incremented in value.

Callback Message Types

Status Messages from DSP Engines

Program Status Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

Update Messages from DSP Engines

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Processing Buffer Ready Messages

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

DSPower-HWLib Function Overview

Creating Programs for DSPower-HWLib and DSPower-SPLib

Acquiring Data Using DSPower-HWLib Software

Semaphore Messages from DSP Engines

Semaphore messages are sent by DSP Engines when a pre-specified flag value is set, indicating that a DSP memory semaphore (flag) has changed. Reserved values can be used to indicate that semaphore

ready messages should be generated by a DSP Engine as a result of native acquisition, digital oscilloscope, or other operating modes which have been initialized and started by the calling application. They can also be generated by user-defined DSP code using either polled or interrupt-driven notification. An example of the former might be a "done", or "not busy" status for DSP accelerated processing.

Semaphore messages are set up when an application calls DSWaitForFlag. Below is a source code example:

```
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG, hwnd);
DSWaitForFlag(hBoard, nFlagNum, dwFlagAddr, uNotifyType);
```

In the above calls, hwnd is first registered to receive buffer ready and semaphore ready messages from the DSP Engine; in this case, hwnd will receive WM_DSPENGINE_FLGRDY messages (see below). hBoard is a handle to a DSP/analog hardware type created by DSAssignBoard, nFlagNum is the number of the flag to wait for, dwFlagAddr the address of the flag's location in DSP board memory, and uNotifyType is the type of notification for the engine to use.

For example, if a flag was initialized to zero, then a DSWaitForFlag call could be used to cause the DSP Engine to send a WM_DSPENGINE_FLGRDY message when the value of the flag became non-zero.

nFlagNum values 0 through 15 are reserved for standard DSP Engine high-level function "semaphore nodes", which are pre-defined status indicators for standard types of analog input and output data and acceleration processing. In these cases, dwFlagAddr and uNotifyType are ignored. For example, an nFlagNum value of 0 indicates accelerated file-based time domain data processing. uNotifyType is ignored for reserved nFlagNum values because DSP Engines may use specialized methods of flag-checking, which may involve hardware register bits or other optimized technique (not necessarily DSP board memory). The concept behind reserved semaphore nodes is that more than one calling application may register a window handle to receive the same semaphore messages.

Values for nFlagNum not equal to 0 through 15 can be used to set up user-defined semaphore messages. In these cases, dwFlagAddr and uNotifyType must be specified. Note that if the initial value of the semaphore should not be zero, then a DSPutMem call should be performed before DSWaitForFlag to initialize the flag value correctly.

NOTE: In the case of user-defined semaphore messages, flag values and memory locations used must be "agreed to" beforehand by the calling application and the executing DSP code.

Currently, values for uNotifyType can be DS_WFB_POLLED and DS_WFB_INT, which specify polling and interrupt-driven notification, respectively. If a uNotifyType value of DS_WFB_POLLED is specified, then the DSP Engine will periodically check to see if the specified semaphore is ready, and cause WM_DSPENGINE_FLGRDY messages to be sent to hwnd as needed. Currently, if uNotifyType is DS_WFB_POLLED, then only one DSWaitForFlag call can be active at one time for each active DSP Engine. To establish multiple DSWaitForFlag requests at one time in polled mode, more than one DSP Engine handle should be active; i.e. more than one copy of .the DSP Engine open simultaneously.

If a uNotifyType value of DS_WFB_INT is specified, the DSP Engine will set up and enable the next available DSP-to-host PC interrupt. When the interrupt occurs, a WM_DSPENGINE_FLGRDY message is sent to hwnd. Note that if uNotifyType is DS_WFB_INT, then more than one DSWaitForFlag call can be established at one time for each active DSP Engine. Note also that for user-defined semaphores using DS_WFB_INT, the dwFlagAddr location is checked to make sure it has changed from its initial value.

The current semaphore messages in DSPower-HWLib and DSPower-SPLib software are listed below:

Semaphore Message	**Summary of Meaning**	**wParam**	**lParam**
WM_DSPENGINE_FLGRDY	DSP code semaphore has changed value	engine identifier	pointer to flag infor-mation

The lParam parameter points to a DSPDATARDYMSGINFO structure, which contains information about the semaphore, including flag number, board handle, and flag address and value. The calling application does not need to return a specific value after processing semaphore messages.

For detailed information about the above messages, see Signalogic DSP Software Reference Guide section 6.1, DLL Messages.

See the enmgr.h file for the current actual value of these messages. Normally, WM_DSPENGINE_FLGRDY is defined as WM_USER (defined in windows.h), and subsequent messages are incremented in value.

Callback Message Types

Status Messages from DSP Engines

Program Status Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

Update Messages from DSP Engines

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Processing Buffer Ready Messages

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

DSPower-HWLib Function Overview

Creating Programs for DSPower-HWLib and DSPower-SPLib

Acquiring Data Using DSPower-HWLib Software

Update Messages from DSP Engines

In DSPower-HWLib, update messages are associated with high-level functions of which the engine is capable. For these functions, the calling application has instructed the DSP Engine to perform and complete--unaided--a complicated task, such as acquiring a waveform file, with only initial instructions being given. Update messages are typically sent when a parameter or other data value has been generated or updated by the DSP Engine as a result of the high-level function which has been invoked by the calling application. Examples include the maximum amplitude in a buffer of acquisition data, a dBA value from a spectrum analyzer function, or current a sample counter in a data logging operation. Some update messages may be sent one-time or sporadically; others may be sent on a regular basis.

A specific example might be the sampling frequency used by the DSP/analog hardware during a DSAcquireWvfrmFile function call. Although a sampling frequency was probably specified in the initial call instructions, the DSP/analog hardware may only allow certain physical values. Once the engine has determined which allowed value is nearest to the desired value, it sends an update message indicating the actual value that will be used. It is up to the calling application to do something with this data or discard it.

A window or control handle must be "registered" to one or more update message types using DSRegisterEngineMsgWnd, before messages can be received. Below is a source code example:

There are several possible update messages which can be sent by DSP Engines. Included are error and status messages, messages related to high-level functions, and user-defined messages originating from DSP code. Some of these messages are listed below:

Update Message	Summary of Meaning	wParam	lParam
WM_DSPENGINE_BUFSIZE	Sent by an engine to indicate the size of the DSP buffer it is using in a high-level function.	engine identifier	contains buffer size
WM_DSPENGINE_SAMPFREQ	Sent by an engine to indicate the actual sampling rate it is	engine identifier	contains sampling frequency

	using in a high-level function.		(32-bit IEEE float)
WM_DSPENGINE_BUFNUM	Sent by an engine to indicate its current buffer number during a high-level function.	engine identifier	contains current buffer number
WM_DSPENGINE_LEVEL	Sent by an engine to indicate the absolute maximum amplitude in the most recent buffer of data during a high-level function.	engine identifier	contains amplitude level

For detailed information about the above messages, see Signalogic DSP Software Reference Guide section 6.1, DLL Messages.

See the enmgr.h file for the current actual value of these messages. Normally, WM_DSPENGINE_BUFRDY is defined as WM_USER (defined in windows.h), and subsequent messages are incremented in value.

Callback Message Types

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

Update Messages from DSP Engines

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Processing Buffer Ready Messages

General Topics

Engine Manager Access

Controlling DSP Engines
Executing Functions in DSP Engines
Registering Callback Messages from DSP Engines
Error Handling in Engine Manager
DSPower-HWLib Function Overview
Creating Programs for DSPower-HWLib and DSPower-SPLib
Acquiring Data Using DSPower-HWLib Software

7.5 Error Handling in Engine Manager

The Engine Manager error handling scheme is straightforward. Most calls return a non-zero value on success, and a zero (NULL) on failure. When calls into enmgr.dll fail, there are two levels of error possibilities: the DLL itself (or "base level"), and engine-related. The DSGetEngMgrErrorStatus and DSGetEngineErrorStatus calls can be used to attempt to learn more information about the most recent failure.

Engine Manager also maintains a "status listbox" which can be made visible using the DSShowEngMgrStatusWindow call. This status window shows a history of *both status and error* text displays resulting from all Engine Manager calls.

Related Topics

General Topics

Engine Manager Access
Controlling DSP Engines
Executing Functions in DSP Engines
Registering Callback Messages from DSP Engines
Error Handling in Engine Manager

7.5.1 DLL Error Handling in Engine Manager

Engine Manager has a "base level" error/status routine which can be used to look for general DLL error conditions not necessarily associated with a specific engine. Below is an example showing further investigation after a DSEngineOpen failure:

```
hEngine = DSEngineOpen(szEnginePath, szWaveformPath);
if (!hEngine) {
    nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}
```

In the above example, the actual return value of DSEngineOpen would be NULL in the case of failure, since a non-NULL value would be a valid engine handle. Thus, to find out more information, the DSGetEngMgrErrorStatus call is used.

Below is a partial list of error values returned by DSGetEngMgrErrorStatus, along with a brief description of their meaning. These constants can be found partly in the enmgr.h file and partly in the dllerr.h file; for a complete list all DSPower series software error conditions by error number, see Error and Status Condition Numeric List and Cross Reference, in Signalogic DSP Software Reference Guide chapter 7, Error and Status Conditions.

dllerr.h or enmgr.h constant	Summary of Meaning
DSCOULDNOTOPENENGINE	Typically means a DSP Engine could not be located.
DSCOULDNOTCREATEBKGNDWINDOW	The background (hidden) window needed by enmgr.dll could not be created.
DSENGINETIMEOUT	Time-out limits were exceeded in waiting for the DSP Engine to successfully open.
DSCOULDNOTSTARTBKGNDTASK	The background (hidden) task needed by enmgr.dll could not be started.
DSNOBKGNDWINDOW	The command failed because there was no background

	window.
DSCOULDNOTINITIALIZEDLL	Enmgr.dll itself could not be initialized.

Note that in C/C++ programs, it is safest to treat the above error definitions as integer types, since some values are "positive" and some "negative". See Error and Status Condition Numeric List and Cross Reference, in Signalogic DSP Software Reference Guide chapter 7, Error and Status Conditions, for more information about this grouping.

Related Topics

DLL Error Handling in Engine Manager

Engine Error Handling

Error/Status Window in Engine Manager

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

7.5.2 Engine Error Handling

When calls into enmgr.dll, hwlib.dll, or splib.dll which directly or indirectly involve an assigned DSP Engine handle fail, DSGetEngineErrorStatus can be used to attempt to learn more information about the failure. This approach allows calls to be treated with logical (boolean) tests inside C/C++ if statements; for example:

```
if (!hBoard = DSAssignBoard(hEngine, szBoardDesignator, uBusType,
wIOBaseAddr, wMemBaseAddr)) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
```

```
}
...continue with valid hBoard...
```

In the above example, if DSAssignBoard is successful, then the hBoard parameter will contain a non-zero value representing the newly assigned board handle. Note that the "base-level" DSGetHWLibErrorStatus routine could also be checked, in case the problem *was not* engine related.

Below is another example, using the DSInitBoard call found in DSPower-HWLib (hwlib.dll):

```
if (!DSInitBoard(hBoard)) {

    if (DSGetEngineErrorStatus(hEngine) ==
    DSHARDWARENOTRESPONDING) {

        MessageBox(NULL, "DSP/analog hardware not responding; are I/O base
        address jumpers set correctly?", "Cool DSP Application #1", MB_OK |
        MB_ICONEXCLAMATION);
    }
}
```

Note that in the above example, an engine handle (hEngine) was implied, since a successful DSAssignBoard call must have been made before attempting a DSInitBoard call.

Below is a partial list of error values returned by DSGetEngineErrorStatus, along with a brief description of their meaning. These constants can be found in the enmgr.h file; for a complete list all DSPower series software error conditions by error number, see Error and Status Condition Numeric List and Cross Reference, in Signalogic DSP Software Reference Guide chapter 7, Error and Status Conditions.

enmgr.h constant	**Summary of Meaning**
DSCOULDNOTOPENENGINE	Typically means the DSP Engine executable program file could not be located.
DSINVALIDENGINEPATH	An invalid engine path was specified in a DSEngineOpen call.
DSENGINETIMEOUT	Time-out limits were exceeded in waiting for the DSP Engine to successfully open and initialize, or respond to a command or request.

DSCOULDNOTOPENPIPE	The Pipe.386 VxD could not be opened for a DOS DSP Engine. Make sure that pipe.386 is listed in the "386Enh" section of the system.ini file on the x:\windows subdirectory.
DSCOULDNOTSTARTENGINE	The DSP Engine failed to initialize correctly.
DSNOENGINESOPEN	A call was made requiring an engine handle but no engines were open.
DSNOENGINEFOUNDINLIST	A call was made using an engine handle which didn't specify an engine currently being maintained by enmgr.dll.
DSENGINEHANDLEWASNULL	A call was made using a NULL engine handle.
DSENGINEMEMORYDISCARDED	A call was made using an engine handle which pointed to memory already discarded.

Note that in C/C++ programs, it is safest to treat the above error definitions as integer types, since some values are "positive" and some "negative". See Error and Status Condition Numeric List and Cross Reference, in Signalogic DSP Software Reference Guide chapter 7, Error and Status Conditions, for more information about this grouping.

Related Topics

DLL Error Handling in Engine Manager

Engine Error Handling

Error/Status Window in Engine Manager

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

7.5.3 Error/Status Window in Engine Manager

The Engine Manager DLL (enmgr.dll) maintains an error and status listbox window at all times. This window can be shown or hidden at any time using the DSShowEngMgrStatusWindow or DSHideEngMgrStatusWindow calls. The Hardware Manager DLL (hwmgr.dll) and DSPower-HWLib and DSPower-SPLib DLLs (hwlib.dll and splib.dll) also make use of this status listbox, by invoking the DSAddEngMgrStatusLine call. This call is exported from the Engine Manager DLL; its prototype can be made available by including enmgr.h. Calling applications including enmgr.h and adding enmgr.lib to their project file can enable the error and status listbox as a debug tool or for operator feedback purposes, and add status listbox lines as needed. For more information, see sections 4.2.2 and 4.2.3, Include Files for DSPower-HWLib and DSPower-SPLib, and .LIB Files for DSPower-HWLib and DSPower-SPLib, respectively.

If displayed, the Engine Manager error and status listbox will show error and status reports not only from DSPower series DLLs, but also DSP Engines. Error and status reports are mixed together, being added to the listbox in chronological order, as they are received.

The position and size of the status and error listbox window is remembered by enmgr.dll in the "enmgr.ini" file, which is stored on the subdirectory from where enmgr.dll was loaded (normally, this is x:\windows\system, where "x" is the DSPower-HWLib or DSPower-SPLib installation drive.).

When the error and status listbox window is displayed, it is visible to whoever is using the application calling enmgr.dll functions. While this may be appropriate during test and development, it may not be for normal operation. One suggestion might be to enable a non-obvious keyboard switch which causes the listbox to become visible, to allow for greater detail in operator-reported errors.

If visible, the Engine Manager error and status listbox can be sized and scrolled like any listbox. All error and status reports added to the listbox are saved, and older reports can be scrolled into view as

needed. Engine Manager makes no attempt to monitor the number of status lines added to the list box, so eventually the oldest previous lines will "scroll off".

See Also

Error/Status Window in Engine Manager: Adding a Line

Error/Status Window in Engine Manager: Showing/Hiding

Related Topics

DLL Error Handling in Engine Manager

Engine Error Handling

Error/Status Window in Engine Manager

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

DLL Error Handling in Engine Manager

Engine Error Handling

Error/Status Window in Engine Manager: Adding a Line

To add a line to the error and status listbox, use the following:

```
hwndStatus = DSAddEngMgrStatusLine(szText);
```

where szText is a null-terminated string (LPCSTR) and hwndStatus is the handle of the status window. Remember the line is added whether the listbox is visible or not.

See Also

Error/Status Window in Engine Manager: Adding a Line

Error/Status Window in Engine Manager: Showing/Hiding

Related Topics

DLL Error Handling in Engine Manager

Engine Error Handling

Error/Status Window in Engine Manager

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

Error/Status Window in Engine Manager: Showing/Hiding

The default state of the Engine Manager error/status listbox is hidden. To make the listbox visible, use:

```
DSShowEngMgrStatusWindow();
```

and to make it not visible, use:

```
DSHideEngMgrStatusWindow();
```

Note that initially the error and status listbox is displayed on top of other windows, but it does not have the "always-on-top" (WS_EX_TOPMOST) attribute, so it can easily be covered by operator activity. If you need the listbox to always stay on top, make the following Windows calls:

```
SetWindowLong(hwndStatus, GWL_EXSTYLE, GetWindowLong(hwndStatus,
GWL_EXSTYLE) | WS_EX_TOPMOST);
```

```
UpdateWindow(hwndStatus);
```

where hwndStatus is the handle of the error and status listbox returned from a DSAddEngMgrStatusLine call.

See Also

Error/Status Window in Engine Manager: Adding a Line

Error/Status Window in Engine Manager: Showing/Hiding

Related Topics

DLL Error Handling in Engine Manager

Engine Error Handling

Error/Status Window in Engine Manager

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager